Union Find Cheatsheet (Java)

Use this algorithm when you have to perform a combination of these queries:

- Combine two nodes/add an edge (or remove an edge, to remove an edge look at the queries in reverse order)
- Determine if two nodes are combined
- Determine the number of nodes a particular node is connected to

Base Code - O(N) per query:

```
class Main {
  public static int [] parent; // initialize parent[i] = i

  public static int findRoot (int a) {
    if (parent[a] == a) {
        return a;
    }
    return parent[a] = findRoot(parent[a]);
  }

  public static boolean isConnected (int a, int b) {
    return findRoot(a) == findRoot(b);
  }

  public static void  join (int a, int b) {
    parent[ findRoot(a) ] = findRoot(b);
  }
}
```

Path Compression - O(log N) per query:

```
public static int findRoot (int a) {
  if (parent[a] == a) {
    return a;
  }
  return parent[a] = findRoot(parent[a]);
}
```

Keep the subtree size small - O(log N) per query:

```
class Main {
  public static int [] size; // initialize size[i] = 1

  public static void  join (int a, int b) {
    a = findRoot(a);
    b = findRoot(b);

    if (a == b) {
        return;
    }

    if (size[a] < size[b]) {
        parent[a] = b;
        size[b] += size[a];
    }
    else {
        parent[b] = a;
        size[a] += size[b];
    }
  }
}
```

Keep the tree depth small - O(log N) per query, a little faster in practice than keeping the subtree size small:

```
class Main {
  public static int [] depth; // initialize depth[i] = 1

  public static void  join (int a, int b) {
    a = findRoot(a);
    b = findRoot(b);

    if (a == b) {
        return;
    }

    if (depth[a] < depth[b]) {
        parent[a] = b;
    }
    else {
```

```
        parent[b] = a;
        depth[a] = max(depth[a], depth[b] + 1);
    }
  }
}
```

Deciding which implementation to use:

If you just have to combine two nodes and query if two nodes are connected: Use path compression (fastest to implement) → O(log N) per query
If you have to query the size a component: Keep the subtree size small → O(log N) per query
If you have to make the runtime super fast: Use path compression and keep the tree depth small → O(log* N) per query, which is almost constant